Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Севастопольский государственный университет»

ОТЧЕТ

о выполнении лабораторной работы № 3 по дисциплине

«Теория принятия решений»

Выполнил:

ст. гр. ИС/б-22-1-о

Гюнтер М. Ю.

Проверил:

Профессор кафедры

“Информационные системы”

Кротов К. В.

Севастополь,2025

# Цель работы: исследовать применение аппарата теории многомерной полезности при принятии решений по выбору эффективных альтернатив.

# Постановка задачи

Задача состоит в выборе одной из альтернатив, представляющих собой выставленные на продажу автомобили. Критериями (характеристиками) решений являются: и . Используя метод, реализующий построение и исследование двумерной функции полезности, для заданных диапазонов значений критериев и их (критериев) дискретных оценок выполнить: формирование линий безразличия , определение на их основе дискретных значений оценок одномерных функций полезности для каждого из критериев и , аппроксимацию дискретных значений одномерных функций полезности с использованием полиномов второй степени, вычисление коэффициента масштабирования j на основе выбираемых ЛПР по кривым безразличия решениям. С использованием сформированных промежуточных решений выполнить для задаваемых характеристик альтернатив вычисление значений одномерных функций полезности, двумерной функции полезности и реализовать выбор эффективного решения. Выполнить вывод исходных данных, всех промежуточных и конечных результатов. Исходными данными для решаемой задачи являются: параметр "цена" изменяется в диапазоне , параметр "пробег" в диапазоне . Шаг дискретизации первого параметра задан равным 25, шаг дискретизации второго параметра задан равным 20. Соответственно, для первого критерия диапазон изменения его значений задан в виде , для второго критерия диапазон задан в виде . Выбор двух эквивалентных решений на одной из кривых безразличия, сформированных программно, выполнить самостоятельно. Данные, на основании которых выбирается эффективное решение, имеют значения, приведенные в таблице ‎2.1.

Таблица ‎2.1 – Данные для выбора эффективного решения

|  |  |  |
| --- | --- | --- |
| Вариант | Цена | Пробег |
| 1 | 30 | 45 |
| 2 | 50 | 30 |
| 3 | 80 | 20 |
| 4 | 25 | 55 |

При выполнении работы необходимо:

# 2.1. Для введенных диапазонов изменения параметров решений (критериев решений) и соответствующих значений этих критериев реализовать процедуру построения двумерной функции полезности , в которой выполнить определение дискретных значений одномерных функций полезности и для соответствующих критериев (реализовать процедуру формирования значений и .

# 2.2. Выполнить построение линий безразличия для двумерной функции полезности , которые в дальнейшем будут использоваться для определения эквивалентных решений, лежащих на одной из этих линий. Координаты этих решений будут использованы при вычислении коэффициента масштабирования .

# 2.3. Реализовать процедуру аппроксимации полученных дискретных значений одномерных функций полезности и с использованием полиномов второй степени ; , результатом реализации этой процедуры являются коэффициенты этих аналитических кривых , , , .

# 2.4. Выполнить формирование процедуры вычисления значения коэффициента масштабирования j, при реализации которой используются координаты и соответствующих эквивалентных решений и , лежащих на одной кривой безразличия (т.е. в качестве исходных данных для этой процедуры использованы координаты , решений и , выбранных на одной кривой безразличия, сформированной в пункте 2.2.

# 2.5. Для задаваемых в варианте характеристик решений с использованием определенных ранее (процедурой в пункте 2.3) аналитических функций ; реализовать процедуру вычисления значений одномерных функций полезности и , а затем двумерной функции полезности с учетом коэффициента масштабирования . В разрабатываемой процедуре выполнить определение эффективного решения с максимальным значением двумерной функции полезности (передаваемыми в реализуемую процедуру наряду с исходными данными являются параметры , , , .

# 2.6. Выполнить вывод: а) линий безразличия, б) полученных значений одномерных функций полезности и , в) видов аппроксимирующих функций ; , г) значений одномерных и двумерной функций полезности для решений, указанных в варианте задания, д) эффективных решений с максимальным значением двумерной функции полезности .

# Ход работы

## Построение линий одинаковой полезности аналитическим путем

По заданному алгоритму построим линии одинаковой полезности:

1. Дискретно заданы значения критериев : 0.01, 0.013, 0.02, 0.04 и : 0.0125, 0.017, 0.025, 0.05.
2. Пусть и – наименьшие значения оценок соответствующих критериев и : 0.01 и 0.0125. Тогда зададим для решения с координатами (, ) значения двумерной и одномерной функций полезности как .
3. Выберем значение критерия , которое больше рассмотренного значения : . Зададим . Отсюда для решения , соответствующего точке критериального пространства , значение многомерной функции полезности . Таким образом получим первую кривую безразличия .
4. Определим значение критерия такое, что , то есть решение , соответствующее точке критериального пространства , эквивалентно известному решению , то есть точки лежат на одной кривой безразличия . Таким значением выберем значение критерия . Поскольку значение многомерной функции полезности , получаем .
5. Для точки критериального пространства , соответствующей решению , определены значения одномерной функции полезности по каждому из критериев: Отсюда значение аддитивной многомерной функции полезности . Получаем, что точка критериального пространства лежит на кривой безразличия .
6. Определим значения критериев и такие, что , то есть решения и , соответствующие точкам критериального пространства и , эквиваленты известному решению , то есть точки лежат на одной кривой безразличия . Такими значениями выберем следующие дискретные значения критериев и 0.02 и 0.025. Поскольку , то .
7. Для точек критериального пространства и , соответствующих решениям и , определены значения одномерной функции полезности по каждому из критериев: . Также выполняется условие соответсвенных замещений, отсюда , и . Получаем, что точки критериального пространства и лежат на кривой безразличия .
8. Определим значения критериев и такие, что , то есть решения и , соответствующие точкам критериального пространства и , эквиваленты известным решениям и , то есть точки лежат на одной кривой безразличия . Такими значениями выберем следующие дискретные значения критериев и 0.04 и 0.05. Тогда , то есть .
9. Для точек критериального пространства , и , соответствующих решениям , и , определены значения одномерной функции полезности по каждому из критериев: . Также выполняется условие соответсвенных замещений, отсюда , и . Получаем, что точки критериального пространства , и лежат на кривой безразличия .
10. Для точек критериального пространства и , соответствующих решениям и , определены значения одномерной функции полезности по каждому из критериев: . Также выполняется условие соответсвенных замещений, отсюда , и . Получаем, что точки критериального пространства и лежат на кривой безразличия .
11. Для точки критериального пространства , соответствующей решению , определены значения одномерной функции полезности по каждому из критериев: . Очевидно, что . Получаем, что точка критериального пространства лежит на кривой безразличия .

Отсюда эффективным решением является решение , имеющее наибольшее значение функции многомерной полезности.

Приведём построенные линии одинаковой полезности на рисунке ‎3.1.

Изображение выглядит как текст, диаграмма, линия, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок ‎3.1 – Полученные аналитически линии одинаковой полезности

## Формирование программы построения линий одинаковой полезности и аппроксимации функций одномерной полезности

Была написана программа на языке Java, позволяющая построить линии одинаковой полезности, определить дискретные значения одномерной функции полезности для каждого из критериев и получить аппроксимированные функции одномерной полезности для каждого из критериев (листинг ‎3.1).

Листинг ‎3.1 – Программа, позволяющая построить кривые безразличия и аппроксимировать функции одномерной полезности

package com.example;

import static java.lang.System.out;

import java.util.ArrayList;

import javax.swing.\*;

import java.awt.\*;

import org.apache.commons.math3.fitting.PolynomialCurveFitter;

import org.apache.commons.math3.fitting.WeightedObservedPoints;

abstract class Criterion {

public double value;

public Double oneDimUtility;

protected Criterion(double value) {

this.value = value;

}

@Override

public String toString() {

return String.format("%s", value);

}

}

class K1 extends Criterion {

public K1(double value) {

super(value);

}

}

class K2 extends Criterion {

public K2(double value) {

super(value);

}

}

class UtilityFunction {

private double linearCoef;

private double quadraticCoef;

private double constantTerm;

private String name;

public UtilityFunction(String name) {

this.name = name;

}

private void calcCoefficients(double[] x, double[] y) {

WeightedObservedPoints points = new WeightedObservedPoints();

for (int i = 0; i < x.length; i++) {

points.add(x[i], y[i]);

}

PolynomialCurveFitter fitter = PolynomialCurveFitter.create(2);

double[] coefs = fitter.fit(points.toList());

this.linearCoef = coefs[0];

this.quadraticCoef = coefs[1];

this.constantTerm = coefs[2];

}

public void approximateUtilityFunction(double[] x, double[] y) {

this.calcCoefficients(x, y);

}

public double calcUtility(double value) {

double utility = this.linearCoef \* value + this.quadraticCoef \* Math.pow(value, 2) + this.constantTerm;

return utility;

}

public void outputFunction() {

out.printf("\nАппроксимация для %s:\n", this.name);

out.printf("U(%s) = %.6f + %.6f\*%s + %.6f\*(%s)^2\n", this.name, this.linearCoef, this.quadraticCoef, this.name.toLowerCase(), this.constantTerm,

this.name.toLowerCase());

}

}

class Alternative {

public String name;

public K1 k1;

public K2 k2;

public Alternative(String name, K1 k1, K2 k2) {

this.name = name;

this.k1 = k1;

this.k2 = k2;

}

public double getTwoDimUtility() {

return k1.oneDimUtility + k2.oneDimUtility;

}

@Override

public String toString() {

return String.format("%s (%s, %s)", this.name, this.k1, this.k2);

}

}

class IndifferenceCurve {

public ArrayList<Alternative> alternatives;

public double twoDimUtility;

public void sort() {

this.alternatives.sort(

(a1, a2) -> {

return Double.compare(a1.k1.value, a2.k1.value);

});

}

@Override

public String toString() {

return alternatives.toString();

}

}

class Problem {

private ArrayList<K1> K1\_values;

private ArrayList<K2> K2\_values;

private ArrayList<Alternative> alternatives;

private ArrayList<IndifferenceCurve> curves;

public Problem(double[] K1\_scale, double[] K2\_scale) {

setK1Values(K1\_scale);

setK2Values(K2\_scale);

setAlternatives();

this.curves = new ArrayList<>();

}

public ArrayList<IndifferenceCurve> getCurves() {

return this.curves;

}

public double[] getK1\_scale() {

double[] K1\_scale = new double[this.K1\_values.size()];

for (int i = 0; i < this.K1\_values.size(); i++) {

K1\_scale[i] = this.K1\_values.get(i).value;

}

return K1\_scale;

}

public double[] getK2\_scale() {

double[] K2\_scale = new double[this.K2\_values.size()];

for (int i = 0; i < this.K2\_values.size(); i++) {

K2\_scale[i] = this.K2\_values.get(i).value;

}

return K2\_scale;

}

public double[] getK1\_utilities() {

double[] K1\_utilities = new double[this.K1\_values.size()];

for (int i = 0; i < this.K1\_values.size(); i++) {

K1\_utilities[i] = this.K1\_values.get(i).oneDimUtility;

}

return K1\_utilities;

}

public double[] getK2\_utilities() {

double[] K2\_utilities = new double[this.K2\_values.size()];

for (int i = 0; i < this.K2\_values.size(); i++) {

K2\_utilities[i] = this.K2\_values.get(i).oneDimUtility;

}

return K2\_utilities;

}

private Alternative getAlternative(K1 k1, K2 k2) {

for (Alternative alternative : this.alternatives) {

if (alternative.k1 == k1 && alternative.k2 == k2) {

return alternative;

}

}

return null;

}

private void setK1Values(double[] scale) {

ArrayList<K1> K1\_values = new ArrayList<>();

for (double val : scale) {

K1\_values.add(new K1(val));

}

this.K1\_values = K1\_values;

}

private void setK2Values(double[] scale) {

ArrayList<K2> K2\_values = new ArrayList<>();

for (double val : scale) {

K2\_values.add(new K2(val));

}

this.K2\_values = K2\_values;

}

private void setAlternatives() {

ArrayList<Alternative> alternatives = new ArrayList<>();

int index = 0;

for (K1 K1\_val : this.K1\_values) {

for (K2 K2\_val : this.K2\_values) {

alternatives.add(new Alternative("x" + ++index, K1\_val, K2\_val));

}

}

this.alternatives = alternatives;

}

public ArrayList<Alternative> findEquivalentAlternatives(int targetTwoDimUtility, int count) {

ArrayList<Alternative> equivalentAlternatives = new ArrayList<>();

for (Alternative alternative : this.alternatives) {

if (Math.abs(alternative.getTwoDimUtility() - targetTwoDimUtility) < 1e-6) {

equivalentAlternatives.add(alternative);

if (equivalentAlternatives.size() == count)

break;

}

}

return equivalentAlternatives;

}

private <C extends Criterion> C findNextCriterionValue(ArrayList<C> c\_values) {

for (C c\_value : c\_values) {

if (c\_value.oneDimUtility == null) {

return c\_value;

}

}

return null;

}

private ArrayList<Alternative> findNextCurveAlternatives(IndifferenceCurve curve) {

curve.sort();

ArrayList<Alternative> baseAlternatives = curve.alternatives;

ArrayList<Alternative> newAlternatives = new ArrayList<>();

for (int i = 0; i < baseAlternatives.size() - 1; i++) {

K1 k1 = baseAlternatives.get(i + 1).k1;

K2 k2 = baseAlternatives.get(i).k2;

Alternative alternative = this.getAlternative(k1, k2);

newAlternatives.add(alternative);

}

return newAlternatives;

}

public void computeIndifferenceCurves() {

this.K1\_values.get(0).oneDimUtility = 0.0;

this.K2\_values.get(0).oneDimUtility = 0.0;

K1\_values.get(1).oneDimUtility = 1.0;

K2\_values.get(1).oneDimUtility = 1.0;

IndifferenceCurve initialCurve = new IndifferenceCurve();

initialCurve.alternatives = new ArrayList<>();

initialCurve.alternatives.add(getAlternative(K1\_values.get(1), K2\_values.get(0)));

initialCurve.alternatives.add(getAlternative(K1\_values.get(0), K2\_values.get(1)));

this.curves.add(initialCurve);

IndifferenceCurve curCurve = new IndifferenceCurve();

curCurve.alternatives = this.findNextCurveAlternatives(initialCurve);

IndifferenceCurve nextCurve;

K1 K1\_val = this.findNextCriterionValue(K1\_values);

K2 K2\_val = this.findNextCriterionValue(K2\_values);

while (K1\_val != null && K2\_val != null) {

nextCurve = new IndifferenceCurve();

curCurve.alternatives.add(getAlternative(K1\_val, K2\_values.get(0)));

curCurve.alternatives.add(getAlternative(K1\_values.get(0), K2\_val));

K1\_val.oneDimUtility = K2\_val.oneDimUtility = curCurve.alternatives.get(0).getTwoDimUtility();

nextCurve.alternatives = this.findNextCurveAlternatives(curCurve);

this.curves.add(curCurve);

K1\_val = this.findNextCriterionValue(K1\_values);

K2\_val = this.findNextCriterionValue(K2\_values);

curCurve = nextCurve;

}

while (!curCurve.alternatives.isEmpty()) {

this.curves.add(curCurve);

nextCurve = new IndifferenceCurve();

nextCurve.alternatives = this.findNextCurveAlternatives(curCurve);

curCurve = nextCurve;

}

}

public double computeScalingCoefficient(Alternative xi, Alternative xj) {

double k1i = xi.k1.oneDimUtility;

double k2i = xi.k2.oneDimUtility;

double k1j = xj.k1.oneDimUtility;

double k2j = xj.k2.oneDimUtility;

double scalingCoef = (k2j - k2i) / (k1i - k2i - k1j + k2j);

return scalingCoef;

}

public void outputAlternativesTwoDimUtilities() {

out.println("Значения полезностей решений:");

for (int i = 0; i < this.alternatives.size(); i++) {

out.printf("%s: %f\n", this.alternatives.get(i).toString(), this.alternatives.get(i).getTwoDimUtility());

}

}

public void outputK1OneDimUtilitities() {

out.println("Значения одномерной полезности для дискретных значений критерия K1:");

for (K1 k1\_val : this.K1\_values) {

out.printf("U(%f) = %.2f\n", k1\_val.value, k1\_val.oneDimUtility);

}

}

public void outputK2OneDimUtilitities() {

out.println("Значения одномерной полезности для дискретных значений критерия K2:");

for (K2 k2\_val : this.K2\_values) {

out.printf("U(%f) = %.2f\n", k2\_val.value, k2\_val.oneDimUtility);

}

}

}

class CurvePlotter extends JPanel {

private ArrayList<IndifferenceCurve> curves;

private double maxK1, maxK2;

public CurvePlotter(ArrayList<IndifferenceCurve> curves, double maxK1, double maxK2) {

this.curves = curves;

this.maxK1 = maxK1;

this.maxK2 = maxK2;

setPreferredSize(new Dimension(700, 700));

setBackground(Color.WHITE);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

g2.setStroke(new BasicStroke(2));

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

int w = getWidth();

int h = getHeight();

int margin = 60;

g2.setColor(Color.BLACK);

g2.drawLine(margin, h - margin, w - margin, h - margin);

g2.drawLine(margin, margin, margin, h - margin);

g2.drawString("K1", w - 40, h - 30);

g2.drawString("K2", 25, 40);

double scaleX = (w - 2.0 \* margin) / maxK1;

double scaleY = (h - 2.0 \* margin) / maxK2;

int numTicks = 6;

g2.setFont(new Font("SansSerif", Font.PLAIN, 11));

g2.setColor(new Color(220, 220, 220));

for (int i = 0; i <= numTicks; i++) {

double valX = (maxK1 / numTicks) \* i;

int x = margin + (int) (valX \* scaleX);

g2.drawLine(x, h - margin, x, margin);

double valY = (maxK2 / numTicks) \* i;

int y = h - margin - (int) (valY \* scaleY);

g2.drawLine(margin, y, w - margin, y);

}

g2.setColor(Color.DARK\_GRAY);

for (int i = 0; i <= numTicks; i++) {

double val = (maxK1 / numTicks) \* i;

int x = margin + (int) (val \* scaleX);

int y = h - margin;

g2.drawLine(x, y - 5, x, y + 5);

String label = String.format("%.3f", val);

g2.drawString(label, x - 12, y + 20);

}

for (int i = 0; i <= numTicks; i++) {

double val = (maxK2 / numTicks) \* i;

int x = margin;

int y = h - margin - (int) (val \* scaleY);

g2.drawLine(x - 5, y, x + 5, y);

String label = String.format("%.3f", val);

g2.drawString(label, x - 45, y + 5);

}

Color[] colors = { Color.BLACK, Color.BLACK, Color.BLACK, Color.BLACK, Color.BLACK };

int colorIndex = 0;

for (IndifferenceCurve curve : curves) {

if (curve.alternatives.isEmpty())

continue;

g2.setColor(colors[colorIndex % colors.length]);

colorIndex++;

int prevX = -1, prevY = -1;

for (Alternative a : curve.alternatives) {

int x = margin + (int) (a.k1.value \* scaleX);

int y = h - margin - (int) (a.k2.value \* scaleY);

g2.fillOval(x - 4, y - 4, 8, 8);

if (prevX != -1)

g2.drawLine(prevX, prevY, x, y);

prevX = x;

prevY = y;

}

}

}

public static void showPlot(ArrayList<IndifferenceCurve> curves, double maxK1, double maxK2) {

JFrame frame = new JFrame("Indifference Curves");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new CurvePlotter(curves, maxK1, maxK2));

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

}

public class App {

public static void main(String[] args) {

double[] K1\_scale = new double[] {

0.01, 0.013, 0.02, 0.04

};

double[] K2\_scale = new double[] {

0.0125, 0.017, 0.025, 0.05

};

Problem p1 = new Problem(K1\_scale, K2\_scale);

p1.computeIndifferenceCurves();

ArrayList<Alternative> equivalentAlternatives = p1.findEquivalentAlternatives(1, 2);

double j = p1.computeScalingCoefficient(equivalentAlternatives.get(0), equivalentAlternatives.get(1));

p1.outputAlternativesTwoDimUtilities();

out.println();

p1.outputK1OneDimUtilitities();

out.println();

p1.outputK2OneDimUtilitities();

out.println();

out.printf("Значение коэффициента шкалирования j: %.2f", j);

out.println();

CurvePlotter.showPlot(p1.getCurves(), K1\_scale[K1\_scale.length - 1], K2\_scale[K2\_scale.length - 1]);

ArrayList<Alternative> candidates = new ArrayList<>();

candidates.add(new Alternative("x1", new K1(1/30.0), new K2(1/45.0)));

candidates.add(new Alternative("x2", new K1(1/50.0), new K2(1/30.0)));

candidates.add(new Alternative("x3", new K1(1/80.0), new K2(1/20.0)));

candidates.add(new Alternative("x4", new K1(1/25.0), new K2(1/55.0)));

UtilityFunction K1\_function = new UtilityFunction("k1");

K1\_function.approximateUtilityFunction(p1.getK1\_scale(), p1.getK1\_utilities());

UtilityFunction K2\_function = new UtilityFunction("k2");

K2\_function.approximateUtilityFunction(p1.getK2\_scale(), p1.getK2\_utilities());

K1\_function.outputFunction();

K2\_function.outputFunction();

out.println();

Alternative bestAlternative = null;

double bestUtility = Double.NEGATIVE\_INFINITY;

for (Alternative alternative : candidates) {

double currentUtility = K1\_function.calcUtility(alternative.k1.value)

+ K2\_function.calcUtility(alternative.k2.value);

if (currentUtility > bestUtility) {

bestAlternative = alternative;

bestUtility = currentUtility;

}

}

out.printf("Решение %s имеет наибольшее аппроксимированное значение двумерной функции полезности %f",

bestAlternative.toString(), bestUtility);

}

}

Построенные линии одинаковой полезности приведены на рисунке ‎3.2.

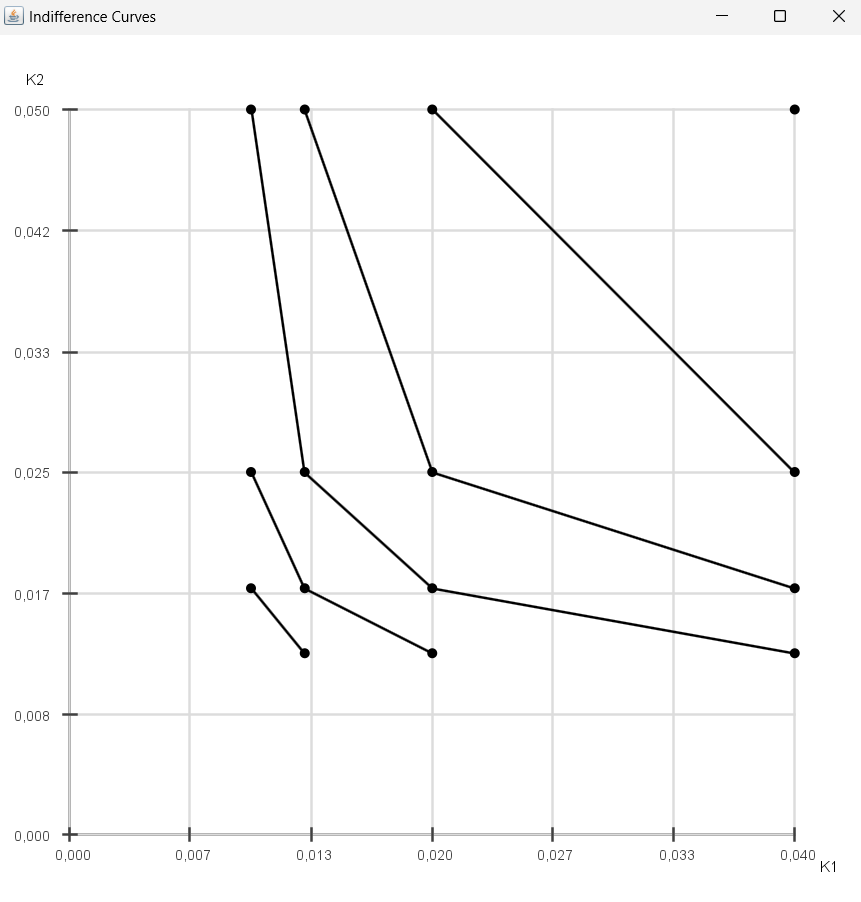


Рисунок ‎3.2 – Построенные программно линии одинаковой полезности

Можно сделать вывод о полном совпадении линий одинаковой полезности, построенных в ходе аналитических расчётов, и линий одинаковой полезности, полученный программно.

Результаты работы программы приведены на рисунке ‎3.3.

Изображение выглядит как текст, снимок экрана, меню, документ

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок ‎3.3 – Результат работы программы

Результаты работы программы говорят о том, что среди заданных решений x1, x2, x3, x4 максимальным решением является со значением двумерной функции полезности -8234,703710.

# Вывод:

В результате выполнения лабораторной работы было сформировано представление о многокритериальном подходе, применяемом для определения эффективных решений. Были рассмотрены понятия замещения по полезности, а также кривые безразличия.